# Socket programmeren

## Inhoudsopgave

* Achtergrond
* Socket API overzicht
* TCP-sockets
* Echo client en server – Echo server – Echo client – De echo client en server uitvoeren – Socket-status weergeven
* Communicatie storing
* Meerdere verbindingen afhandelen
* Client en server met meerdere verbindingen – Server met meerdere verbindingen – Client met meerdere verbindingen – De client en server met meerdere verbindingen uitvoeren
* Toepassingen van client en server – Toepassing protocol header – Een toepassing-bericht verzenden – Application message Klasse
* Probleemoplossing – ping – netstat – Windows – Wireshark
* Bronnen – Python documentatie – Fouten (Errors) – Socket adres features – Hostnamen – Het blokkeren van oproepen – Het afsluiten van verbindingen – Byte Endianness
* Conclusie

## Achtergrond

Sockets en de socket API worden gebruikt om berichten over een netwerk te verzenden. Ze bieden een vorm van interprocescommunicatie (IPC). Het netwerk kan een logisch, lokaal netwerk met de computer zijn of een netwerk dat fysiek is verbonden met een extern netwerk, met eigen verbindingen met andere netwerken. Het voor de hand liggende voorbeeld is het INTERNET, waarmee u verbinding maakt via uw ISP.

## Socket API Overzicht

De socket-module van Python biedt een interface naar de Berkeley Sockets API. Dit is de module die u in deze zelfstudie zult gebruiken.

* socket()
* .bind()
* .listen()
* .accept()
* .connect()
* .connect\_ex()
* .send()
* .recv()
* .close()

Python biedt een handige consistente API die rechtstreeks wordt toegewezen aan systeemoproepen, hun C-tegenhangers (C programming language counterparts). In het volgende gedeelte leert u HOE DEZE SAMEN WORDEN GEBRUIKT.

## TCP Sockets aanmaken

U gaat een socketobject maken. Wanneer u dat doet, is het standaardprotocol dat wordt gebruikt **TCP (Transmission Control Protocol)**. Dit is een professionele-niveau standaard die worden gebruikt bij bedrijven van elk kwaliteit.

* Waarom zou u TCP gebruiken? – Betrouwbaarheid: Pakketten die in het netwerk vallen worden gedetecteerd en opnieuw verzonden door de afzender.
* Datalevering: Gegevens worden door uw toepassing gelezen in de volgorde waarin deze door de afzender zijn geschreven.

**UPD (User Diagram Protocol)** zijn daarentegen niet echt betrouwbaar en gegevens die door de ontvanger worden gelezen, kunnen niet in orde zijn met de schrijfbewerkingen van de afzender.

Maar,

WAAROM is dit BELANGRIJK?

Netwerken zijn een best-effort delivery systeem. Er is GEEN garantie dat uw gegevens hun bestemming (destination) bereiken of dat u ontvangt wat naar u is verzonden.

## Pakketverlies

Netwerkapparaten, zoals routers en switches, hebben en eindige bandbreedte beschikbaar en hebben hun eigen inherente systeembeperkingen. Ze hebben CPU’s, geheugen (RAM), bussen, en interfacepakketbuffers, net als uw client en servers. TCP voorkomt dat u zich zorgen hoeft te maken over *pakketverlies*, out-of-order gegevensinvoer, en andere valkuilen die altijd optreden wanneer u via een netwerk communiceert.

## Client en Server

Hier is de server:

# echo-server.py  
  
import socket  
  
HOST = "127.0.0.1" # Standaard loopback interface adres (localhost).  
PORT = 65432 # Portnummer (integer) om te luisteren (ports zonder privilege zijn groter dan 1023)  
  
# srv = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as srv:  
  
 srv.bind((HOST, PORT))  
 srv.listen()  
  
 conn, addr = srv.accept()  
  
 with conn:  
 print(f"Connected by {addr}")  
 while True:  
 data = conn.recv(1024)  
 if not data:  
 break  
 conn.sendall(data)

Client gaat request sturen en responses ontvangen:

# echo-client.py  
  
import socket  
  
HOST = "127.0.0.1" # Standaard loopback interface adres (localhost).  
PORT = 65432 # Portnummer (integer) om te luisteren (ports zonder privilege zijn groter dan 1023)  
  
with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as clt:  
 clt.connect((HOST,PORT))  
 clt.sendall(b"Hello, world")  
 data = clt.recv(1024)  
  
print(f"Received {data!r}")